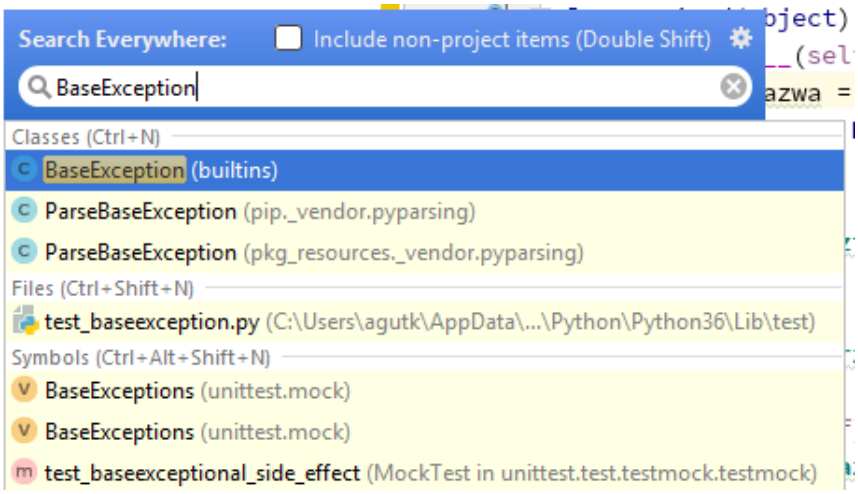
PODSTAWY PROGRAMOWANIA W PYTHON

PO 10 I 11 ZAJĘCIACH:

1. Omawiane zagadnienia:
    a. **klasy** – kontynuacja – definiowane klas, używanie instancji
    b. **klasy –** przeciążanie (overriding) metod i atrybutów specjalnych:
       operatory: +, -, ==, <, >
       zachowanie metod: print, len itp.
       Metody specjalne – dunder (**d**ouble **under**score) podwójny podkreślnik przed i po nazwie: __init__ __str__ __add__ __lt__ __gt__ itp.
    c. **paradygmaty programowania obiektowego:** abstrakcja, enkapsulacja, dziedziczenie, polimorfizm
    d. **klasy – dziedziczenie:**
        i. klasa bazowa (rodzic, nadrzędna) – używa w sygnaturze object,
        ii. klasy dziedziczące (dziecko, podklasa) – w sygnaturze używamy nazwę klasy z której dziedziczymy
        iii. klasa dziedzicząca może być rodzicem dla innej klasy
        iv. z jednej klasy może dziedziczyć wiele klas
        v. jedna klasa może dziedziczyć z wielu klas
        vi. dziedziczenie odbywa się z góry na dół
        vii. podklasa korzysta z implementacji istniejących w klasie nadrzędnej, może też zdefiniować własną implementację
    e. **dziedziczenie diamentowe**:
       w Python dozwolone jest dziedziczenie z wielu klas. Musimy w takiej sytuacja pamiętać o kolejności wyszukiwania atrybutów – Python będzie je wyszukiwał w kolejności w jakiej były zdefiniowane w sygnaturze klasy. Jeśli chcemy użyć metodę/pole z klasy innej musimy się do niej odwołać.
        i. **Method Resolution Order** (MRO) – kolejność w jakiej Python szuka atrybutów w klasach-rodzicach – użycie **help(**NazwaKlasy**)**

f. **pola klasy** (zmienne klasy) – zmienne umieszczone na poziomie klasy, ich zawartość jest widoczna przez wszystkie instancje. Najczęściej używamy je do trzymania tzw. stałych, lub domyślnych wartości. Przy ich **definiowaniu nie używamy** słowa self
    i. obiekt może nadpisać pole klasy - czyli mieć odmienną wartość niż określone w polu klasy.
g. **metody klasy** – oznaczamy dekoratorem **@classmethod**, definiujemy jak metodę instancji, ale zamiast słowa self używamy **cls**. cls oznacza, że jako pierwszy argument, do metody jest przekazywana klasa.
   Metody te używamy w celu manipulowania polami klasy, lub jako alternatywne konstruktory. Jeśli korzystamy z nich jak z konstruktorów to musimy pamiętać o kolejności – tworzymy instancję, zmieniamy dane wg. argumentów i na końcu zwracamy gotowy obiekt.
h. **metody statyczne** – używamy dekoratora **@staticmethod**– metody, które można użyć bez przekazywania klasy lub instancji. Metody te wykorzystujemy w sytuacji gdy jakaś funkcjonalność jest związana z naszym modułem, ale nie jest konieczne tworzenie instancji. Np. wyobraźmy sobie moduł zarządzania pracownikami, możemy mieć w nim metodę statyczną, która będzie sprawdzała, czy numer PESEL jest poprawny, lub w module płatności sprawdzamy czy numer karty jest poprawny – w tym celu nie musimy tworzyć instancji pracownika, ani instancji płatności.
i. **__del__** – usuwanie obiektów/pól – zobaczyliśmy, że w momencie zakończenia programu Python wywołuje destruktor wszystkich obiektów
j. **__dict__** - zawiera słownik, w którym są wszystkie zmienne danego obiektu oraz jego wartości.

2. Kod Python też jest zdefiniowany w ten sam sposób jak tworzymy własne obiekty. Mieliśmy styczność z klasami, które miały wspólne atrybuty, a jednak się różniły – były to wyjątki. Zobacz jak to wygląda w kodzie Python – W PyCharm naciśnij dwa razy klawisz Shift, pojawi się okienko wyszukiwania, wpisz w nim BaseException i naciśnij enter – PyCharm otworzy i przeniesie Cię w odpowiednie miejsce modułu builtins. Zobacz w jaki sposób wyjątki dziedziczą z klasy BaseException.



Możesz też poszukać innych klas dziedziczących (np. NamedTuple)

Dla poćwiczenia – potworzyć klasy, jeśli czujemy się pewniej to możemy przerobić bazę na klasy, dodać pola i metodę klasy i metodę statyczną. Możemy poćwiczyć dziedziczenie.

PS.
Don't be like this guy.